🔧 Manual LoRA Injection for Hugging Face Transformers (Without PEFT)

This repository demonstrates how to apply **LoRA (Low-Rank Adaptation)** to a Hugging Face Transformer model **without using the PEFT library**.
Instead of relying on high-level abstractions, LoRA is injected manually into the attention projection layers, providing **full architectural control, transparency, and experimental flexibility**.

The project uses **DistilBERT (smallest practical HF Transformer)** as a minimal working example.

🎯 Objectives

- Apply LoRA to Hugging Face models **without PEFT**
- Freeze base model weights and train **only low-rank adapters**
- Maintain full control over:
    - where LoRA is injected (Q/K/V)
    - rank, scaling, and merge behavior
- Enable:
    - saving/loading LoRA weights independently
    - merging LoRA into base weights for deployment
- Provide a clean, reproducible research scaffold

🧠 Conceptual Overview

Instead of fine-tuning all parameters of a Transformer, LoRA decomposes weight updates:

$$W' = W + \Delta W, \quad \Delta W = BA$$

Where:

- WWW: frozen pretrained weight
- A,BA, BA,B: low-rank trainable matrices
- Only AAA and BBB are optimized

This dramatically reduces trainable parameters while preserving pretrained knowledge.

🏗️ Architecture

LoRA is injected into the **self-attention projections**:

- Query (Q)
- Key (K)
- Value (V)

Each `Linear` layer is wrapped by a custom `LoRALinear` module:

```
Original:   y = Wx
LoRA:       y = Wx + α · B(Ax)
```

No external adaptation library is used.

📦 Features

- ☑️ Manual LoRA implementation (no PEFT)
- ☑️ Hugging Face ready-to-use model integration
- ☑️ Base model frozen
- ☑️ Trainable parameter count verification
- ☑️ LoRA-only save/load
- ☑️ Merge LoRA into base model for production export
- ☑️ Minimal, readable code

🚀 Quick Start

1 Install Dependencies

```
pip install torch transformers
```

---

2 Run the Experiment

```
python main.py
```

The script will:

- Load DistilBERT
- Freeze base weights
- Inject LoRA into attention layers
- Train only LoRA parameters
- Save LoRA weights
- Reload LoRA
- Merge LoRA into base model
- Run inference

📊 Parameter Efficiency Example

Typical output:

```
Total parameters:         ~66,000,000
Trainable (LoRA only):    ~16,000
Trainable ratio:          ~0.024%
```

This confirms that adaptation happens **without full fine-tuning**.

💾 Saving & Reusing LoRA

```
save_lora_weights(model, "lora.pt")
load_lora_weights(model, "lora.pt")
```

You can attach different LoRA adapters to the same base model for different tasks.

🔗 Merging for Deployment

```
merge_lora_into_base(model)
```

After merging:

- LoRA layers disappear
- Model becomes standard `nn.Linear`
- Ready for ONNX / TorchScript export

🔬 Research & Engineering Use Cases

- LoRA vs Full Fine-tuning benchmarks
- Parameter-efficient transfer learning
- Domain adaptation
- Multi-task adapter switching
- Low-memory GPU environments
- Production deployment without PEFT dependency

⚠️ Important Notes

- This project is **not about achieving SOTA performance**
- It is a **mechanistic and architectural demonstration**
- Designed for:
  - transparency
  - experimental control
  - research reproducibility

🧩 Extending This Work

You can easily extend this scaffold to:

- Other HF models (BERT, RoBERTa, GPT-style)
- Different LoRA ranks per layer
- LoRA on MLP projections
- Multiple LoRA heads per task
- Adapter composition / routing

📚 References

- Hu et al., 2021 – *LoRA: Low-Rank Adaptation of Large Language Models*
- Hugging Face Transformers documentation
- PyTorch `nn.Module` system